C++ pour la robotique

Introduction au C++

Sébastien Rothhut

Présentation & tour de table

- Nom & prénom
- Expérience de code (C, C++, autre)
- Expérience d’Arduino ou autres cartes de développement
- Attentes pour le cours

Objectifs

- Découverte du langage C++
- Premiers programmes
- Programmation de cartes Arduino Uno
- Programmation de NVidia Jetson Nano

Tour d’horizon du C

Langage de programmation

- compilé
- impératif
- bas niveau

1972 : Création du langage

1989 - 1990 : normes ANSI et ISO (C89 - C90)

Évolutions : C99, puis C11

Utilisations : programmation système, embarqué, robotique

Tour d’horizon du C++

Langage :

- compilé
- concepts de programmation impérative, de POO, de programmation générique
- compatible avec le C (sur-ensemble du C)

1979 - 1984 : de “C with classes” à C++

1985 : *The C++ Programming Language* et premier compilateur

1998 : norme ISO (ISO/CEI 14882:1998), évolution C++11 en 2011 puis C++14, C++17, C++20

Utilisations : robotique, embarqué, moteurs de jeux vidéo

Tour d’horizon d’Arduino

Plateforme électronique open source

Développée à l’école de Design d’Interaction d’Ivrea en Italie en 2005, conçue pour être un environnement de prototypage rapide

Carte + environnement de programmation

Programmer en C++ : fichiers source

Suite d’instructions respectant la syntaxe et la grammaire du langage

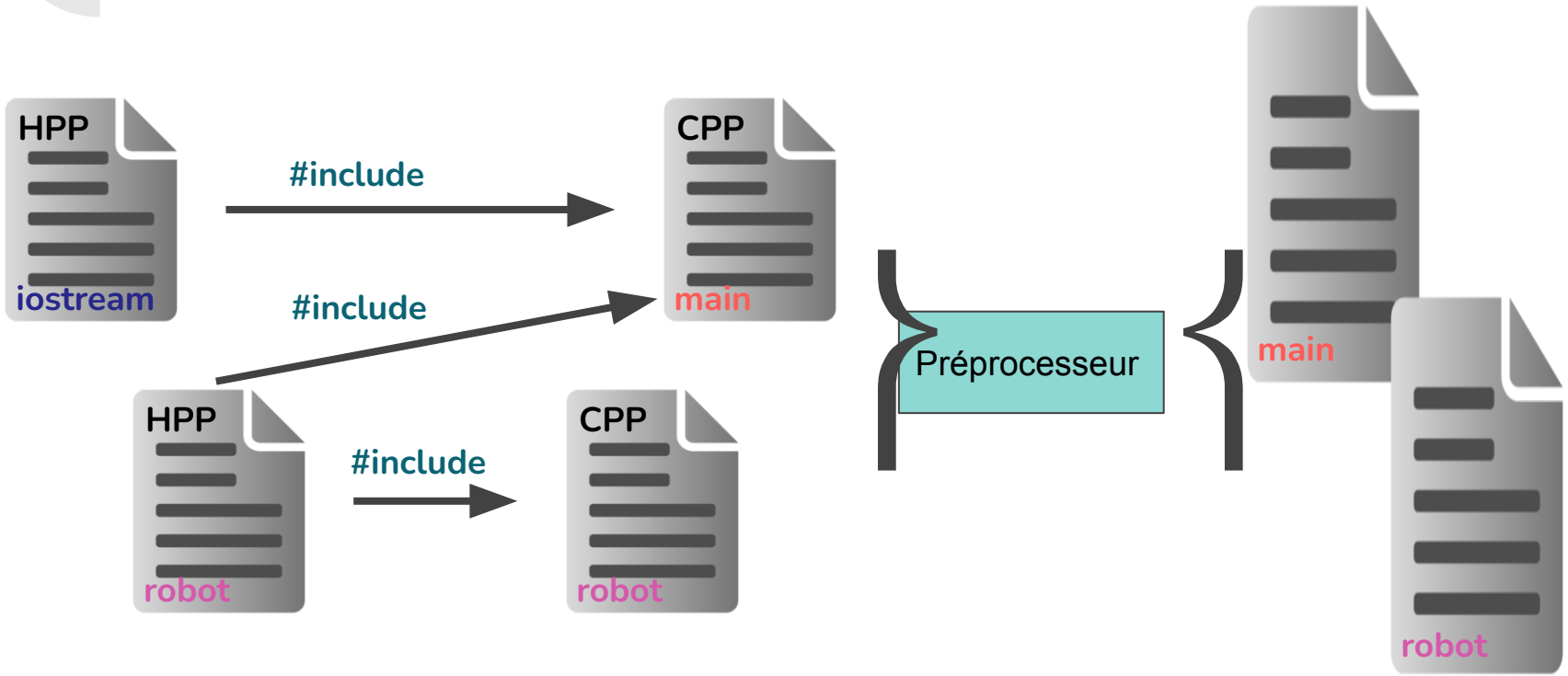
- Fichiers .cpp : instructions
- Fichiers .h ou .hpp (header) : déclarations

Programmer en C++ : fichiers source

Les fichiers .h / .hpp regroupent les définitions nécessaires à l'exécution des instructions contenues dans les fichiers .cpp

→ les fichiers .cpp incluent les fichiers .h / .hpp requis grâce à la directive **#include**

Programmer en C++ : fichiers source



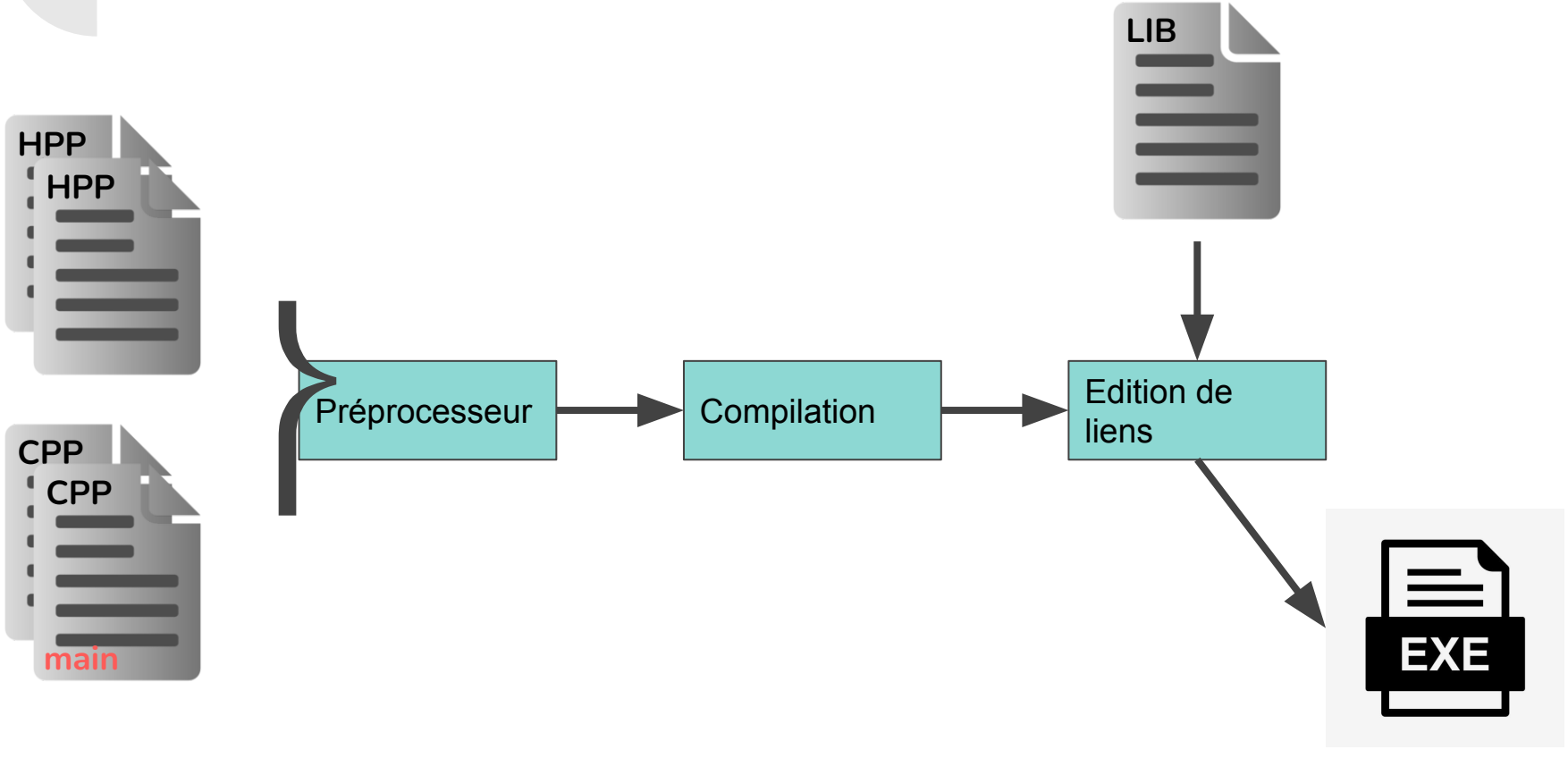
Programmer en C++ : compilation

Un programme C++ doit être **compilé** avant de pouvoir être exécuté.

→ compilation = traduction en langage machine, selon l'architecture cible

Arduino utilise avr-g++, dérivé de g++

Cycle de vie du programme



Premier code C++ : Hello World

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout<<"Hello World";

    return 0;
}
```

Premier code C++ : Hello World

A votre tour :

https://www.onlinegdb.com/online_c++_compiler/

- Exécuter le code du Hello World
- Modifier le code pour afficher “Hello ROBO3”

Structure d’un programme C++

- Liens
  - inclusion de fichiers d’en-tête : #include <iostream>
  - namespaces : using namespace std;
- Déclarations globales : variables, définitions de classes
- Déclaration des fonctions
- Fonction ***main*** dont le prototype* peut être :
  - int main();
  - int main(int argc, char *argv[]);

Cette fonction doit être unique, c’est celle qui sera appelée au démarrage du programme

La syntaxe

```cpp
/*
 * syntax_example.cpp
 *
 *  Created on: sep. 02 2022
 *      Author: Sébastien Rothhut
 */

#include <iostream>

using namespace std;

int i = 1;
int j = i + 1;

int function() {
    if (j > i) {
        cout<<"Hello World";
    } else {
        // On ne devrait jamais arriver ici
        cout<<"Erreur";
    }
    return 0;
}

int main()
{
    function();
    return 0;
}
```

La syntaxe

- commentaires
- déclaration de variables
  - type
  - nom
  - opérateur d'affectation
  - valeur
  - point virgule
- définition d'une fonction
  - type de retour
  - nom
  - paramètres
  - indentation
- instruction if / else
  - expression booléenne
  - opérateur de comparaison
  - blocs d'instructions
- instruction return

```cpp
/*
 * syntax_example.cpp
 *
 *  Created on: sep. 02 2022
 *      Author: Sébastien Rothhut
 */

#include <iostream>

using namespace std;

int i = 1;
int j = i + 1;

int function() {
    if (j > i) {
        cout<<"Hello World";
    } else {
        // On ne devrait jamais arriver ici
        cout<<"Erreur";
    }
    return 0;
}

int main()
{
    function();
    return 0;
}
```

Commentaires

Améliorent la lisibilité du code source

Peuvent être utilisés pour générer une documentation (ex : Doxygen)

```
/*
 * Commentaire
 * sur plusieurs
 * lignes
 */

// Commentaire sur une ligne
```

Commentaires

Commentez, commentez, commentez !

Les variables

Une variable est typée à sa déclaration

Les variables

Déclaration : sans initialisation, la valeur est indéterminée

```
int k;
```

Exercice : déclarer un entier sans l'initialiser et afficher sa valeur

Les variables

Initialisation

```
int k;
k = 1;
```

```
bool b = true;
```

```
float a = 2., b = 5.;
```

Constantes

```
const float pi = 3.14159;
```

Mots-clefs du langage

alignas (C++11)
alignof (C++11)
and
and_eq
asm
atomic_cancel (TM TS)
atomic_commit (TM TS)
atomic_noexcept (TM TS)
auto (1)
bitand
bitor
bool
break
case
catch

char
char8_t (C++20)
char16_t (C++11)
char32_t (C++11)
class (1)
compl
concept (C++20)
const
consteval (C++20)
constexpr (C++11)
constinit (C++20)
const_cast
continue
co_await (C++20)
co_return (C++20)
co_yield (C++20)

decltype (C++11)
default (1)
delete (1)
do
double
dynamic_cast
else
enum
explicit
export (1) (3)
extern (1)
false
float
for
friend
goto
if

inline (1)
int
long
mutable (1)
namespace
new
noexcept (C++11)
not
not_eq
nullptr (C++11)
operator
or
or_eq
private
protected
public

reflexpr (reflection TS)
register (2)
reinterpret_cast
requires (C++20)
return
short
signed
sizeof (1)
static
static_assert (C++11)
static_cast
struct (1)
switch
synchronized (TM TS)
template
this (4)

thread_local (C++11)
throw
true
try
typedef
typeid
typename
union
unsigned
using (1)
virtual
void
volatile
wchar_t
while
xor
xor_eq

Mots-clefs du langage

alignas (C++11)
alignof (C++11)
and
and_eq
asm
atomic_cancel (TM TS)
atomic_commit (TM TS)
atomic_noexcept (TM TS)
auto (1)
bitand
bitor
bool
break
case
catch

thread_local (C++11)
xor
xor_eq

https://en.cppreference.com/w/cpp/keyword

Types primitifs les plus courants

Types numériques :

- **short** : (au moins) 16 bits - signed (par défaut) / unsigned
- **int** : (au moins) 16 bits - signed (par défaut) / unsigned
- **long** : (au moins) 32 bits - signed (par défaut) / unsigned
- **float** : nombre flottant, sur 32 bits
- **double** : nombre flottant, sur 64 bits

Types primitifs les plus courants

Types numériques de longueur fixe:

- **int8_t, int16_t, int32_t, int64_t**
- **uint8_t, uint16_t, uint32_t, uint64_t**

Types primitifs les plus courants

Exercice : Quelle est la valeur du plus grand entier représentable par un `uint16_t` ?

Exercice : Écrire un programme qui initialise un `uint16_t` à la plus grande valeur, lui ajouter 2, et afficher le résultat

Exercice : Écrire un programme qui initialise un `uint16_t` à la plus grande valeur + 2, et afficher le résultat

Types primitifs les plus courants

```cpp
#include <iostream>

using namespace std;

int main()
{
    uint16_t k = 65535;
    k = k+2;
    cout << k;
    return 0;
}
```

```cpp
#include <iostream>

using namespace std;

int main()
{
    uint16_t k = 65535 + 2;
    cout << k;
    return 0;
}
```

Types primitifs les plus courants

```
main.cpp: In function ‘int main()’:
main.cpp:7:24: warning: unsigned conversion from ‘int’ to ‘uint16_t’ {aka
‘short unsigned int’} changes value from ‘65537’ to ‘1’ [-Woverflow]
     7 |     uint16_t k = 65535 + 2;
       |                  ~~~~~~^~~
```

Types primitifs les plus courants

Caractères : **char**

Booléens : **bool** (**true** / **false**)

Type vide : **void**, utilisé pour les fonctions qui ne retournent aucune valeur

Opérateurs

Affectation :

- i = 3;
- i += 3; // équivaut à i = i+3

Opérateurs

Entrée / Sortie, en conjonction avec les flux **cin** et **cout** définis dans la bibliothèque iostream

- `std::cin >> x; // Affecte à x la valeur tapée sur l'entrée standard`
- `std::cout << "Message à afficher" << endl; // Affiche le message sur la sortie standard`

Exercice

Écrire un programme qui demande votre nom en entrée et affiche en sortie “My name is <votre nom>”

Pour stocker une chaîne de caractères, utiliser le type `string`, fourni par la bibliothèque `string`

Exercice

```cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string name;
    cout << endl << "Entrez votre nom : ";
    cin >> name;
    cout << "My name is " << name;
    return 0;
}
```

Opérateurs

Opérateurs booléens :

- Négation : **!x**
- Ou : **x || y**
- Et : **x && y**

Opérateurs

Opérateurs arithmétiques : +, -, *, /, %

Incrémentation : `i++`;
Après l'exécution de cette instruction, i sera incrémenté de 1

Décrémentation : `i--`;

Opérateurs

Expressions booléennes :

- Comparaison : ==, !=, >, <, >=, <=
- **true** / **false**
- Toute valeur != 0 vaut **true**, toute valeur == 0 vaut **false**

Exercice

Écrire un programme qui demande à l'utilisateur de taper 5 entiers et qui affiche leur moyenne. Le programme ne devra utiliser que 2 variables.

Exercice

```cpp
#include<iostream>
using namespace std;
int main() {
     int a;
     double s=0;

     cout<<"Tapez la valeur numero 1 : ";
     cin>>a;
     s=s+a;
     cout<<"Tapez la valeur numero 2 : ";
     cin>>a;
     s=s+a;
     cout<<"Tapez la valeur numero 3 : ";
     cin>>a;
     s=s+a;
     cout<<"Tapez la valeur numero 4 : ";
     cin>>a;
     s=s+a;
     cout<<"Tapez la valeur numero 5 : ";
     cin>>a;
     s=s+a;

     s=s/5.0;
     cout<<"La moyenne vaut : "<<s<<endl;

     return 0;
}
```

Blocs de code et visibilité

Un bloc de code est délimité par des accolades

Il définit la **portée** des variables qui y sont définies

```
int m = 1, n = 2;
{
    int o = m + n;
    m = 5;
}

o = 6;
```

Blocs de code et visibilité

Ici, m et n sont déclarées à l'extérieur du bloc, elles sont visibles à l'intérieur

A contrario, o est invisible à l'extérieur du bloc dans lequel elle a été définie

```
int m = 1, n = 2;
{
    int o = m + n;
    m = 5;
}

o = 6;
```

Blocs de code et visibilité

Une variable définie à l'extérieur de tout bloc est **globale**

```
int m = 1, n = 2;
{
    int o = m + n;
    m = 5;
}

o = 6;
```

Structures de contrôle

Instruction conditionnelle **if** / **else**

```
if (i > 10) {
    // Instructions
} else {
    // Instructions
}
```

Structures de contrôle

Instruction conditionnelle **if** / **else**

```
if (i > 10) {
    // Branche d'exécution
} else if ( i <= 10 && i > 5) {
    // Autre branche
} else {
    // Cas par défaut
}
```

Structures de contrôle

```
if (i > 10) {
    // Branche d'exécution
} else if (i == 10) {
    // Autre branche
} else if (i == 9) {
    // Autre branche
} else if (i == 8) {
    // Encore une autre branche
} els
    /
} els
    /
} els
    /
}
```

Structures de contrôle

Instruction conditionnelle
switch / case

- L'instruction **break** interrompt le **switch**
- Le bloc **default** intercepte tous les cas non traités

```
switch (i) {
    case 10:
    case 9:
    case 8:
        cout<<"i >= 8"<<endl;
        break;
    case 7:
    case 6:
    case 5:
        cout<<"i entre 5 et 7"<<endl;
        break;
    default:
        cout<<"i < 5"<<endl;
        break;
}
```

Structures de contrôle

Boucle **while**

```
int cpt = 0;
while (cpt < 10) {
    cout << "cpt vaut " << cpt << endl;
    cpt++;
}
```

Structures de contrôle

Boucle **for**

```
for (int cpt = 0; cpt < 10; ++cpt) {
    cout << "cpt vaut " << cpt << endl;
}
```

Structures de contrôle

Instruction **break**

```
int input;
for (int cpt = 0; cpt < 5; ++cpt)
{
    cin >> input;
    if (input == 42) {
        cout << "gagné" << endl;
        break;
    }
}
```

Lisibilité - maintenabilité

- Indentation cohérente
- Une instruction par ligne
- Noms de variables et de fonctions explicites
- Pas de “nombres magiques”
- Commentaires (j’insiste !)

Exercice

Modifier le programme de l’exercice précédent afin d’utiliser une boucle pour calculer la moyenne des 5 entiers.

Exercice

```
#include<iostream>
using namespace std;
int main() {
    int a;
    double s=0;

    for(int i = 1; i <= 5; ++i) {
        cout<<"Tapez la valeur numero "<< i <<" : ";
        cin>>a;
        s=s+a;
    }

    s=s/5.0;
    cout<<"La moyenne vaut : "<<s<<endl;
    return 0;
}
```

Exercice

Écrire un programme qui demande à l’utilisateur de taper un entier N entre 0 et 20 bornes incluses et qui affiche N+17.

Si on tape une valeur erronée, il faut afficher "erreur" et demander de saisir à nouveau l'entier.

Exercice

```
#include<iostream>
using namespace std;

int main() {
     int nn;

     while (1)  {
          cout<<"Entrer un entier entre 0 et 20 inclusif :";
          cin >> nn;
          if ((nn>=0) && (nn<=20)) break;
          cout << " erreur\n";
     }
     cout << "\n";

     cout << nn << " + 17 = " << nn+17;
     return 0;
}
```

Devoirs maison

Pour le prochain cours...

- Télécharger Virtual Box

https://www.oracle.com/fr/virtualization/technologies/vm/downloads/virtualbox-downloads.html?source=:ow:o:p:nav:mmddyyVirtualBoxHero_fr&intcmp=:ow:o:p:nav:mmddyyVirtualBoxHero_fr

Devoirs maison

Pour le prochain cours...

- Télécharger Xubuntu

https://xubuntu.org/download/

(Optionnel) Eclipse CDC - PlatformIO

https://www.eclipse.org/downloads/packages/release/2022-06/r/eclipse-ide-cc-developers

The CDT can be installed into an existing Eclipse using the "Install New Software..." dialog

https://www.eclipse.org/cdt/doc/overview/#/

https://docs.platformio.org/en/stable/integration/ide/eclipse.html

https://docs.platformio.org/en/stable/core/installation/methods/installer-script.html#local-download-macos-linux-windows

/!\ Besoin de Python >= 3.6